DSA Homework 3: Stacks and Queues

David Tarazi

February 14, 2020

1 Queue With 2 Stacks

You can use two stacks: stack1 contains the current stack and to enqueue you simply push to stack1. To dequeue, you need stack2 to copy over the current stack and to retrieve the last element in the stack to pop. You throw an error if both stacks are empty, but otherwise you iterate through stack1 and push each element to stack2 until stack1 is empty. Then you pop the last element in stack2. You then have successfully preserved the queue while still being able to dequeue/pop. Now that stack2 has all the elements, you just need to check which stack is empty to copy everything over.

This way, an enqueue is the same as a push making the operation O(1) runtime. However, to dequeue you must push (O(1)) n times then pop once (O(1)). Therefore, given a list with at least one element, $1 \leq pushes \leq n$ making this operation

$$\frac{O(1) * n + O(1)}{n_{iterations}} = O(1)amortized \tag{1}$$

2 Deque Runtime

Utilizing the bankers method for amortized analysis, the cost of each operation is listed below:

- push front = 1
- push back = 1
- pop front = n/2 (current size)
- pop back = n/2 (current size)

Essentially, when you pop front or back, you assign n/2 cost to that operation so that you have n/2 free operations which would be enough to freely restructure the deque if needed. There will never be a pop that costs more than the number of elements in the deque (n) and there will never be more "expensive" pops than "cheap" pops so you can cut the cost to n/2 as opposed to n. Therefore the total cost, C, given all operations is shown below.

$$C = \frac{(2 * \frac{n}{2} + 2 * 1) * n}{n} \tag{2}$$

The amortized runtime is therefore O(n) when you divide through by n iterations.

3 Queue Implementation and Runtime

See Python code at dtarazi.py
To analyze runtime, there are a few functions to keep in mind and their cost using the potential function method to find the amortized cost. The individual cost for each function is shown below.

- Enqueue: $(1) + (n + 1) - (n) = 2 = O(1)$

- Dequeue: $(1) + (n-1) - (n) = 0 = O(1)$
- Find Minimum (worst case): $(n) + (n) - (n) = n = O(n)$

When looking at all of these combined, it is clear that the runtime for the queue I have constructed is O(n).